Capítulo SETE

Coleções

Objetivos do Exame

Criar e usar objetos ArrayList, TreeSet, TreeMap e ArrayDeque.

Visão Geral de Coleções

Uma **coleção** é um termo genérico que se refere a um **contêiner de objetos**.

O **Java Collections Framework** é uma biblioteca de classes e interfaces no pacote java.util que fornece coleções com diferentes características.

As interfaces mais importantes são:

- **Collection**
  Esta é a interface base da hierarquia de coleções e contém métodos como
  add(), remove(), clear() e size().
- **Iterable**
  Implementar esta interface permite que um objeto seja "iterável" com um loop for-each, através de um Iterator e com o novo método forEach().
- **List**
  Interface para coleções que: (1) armazenam um grupo de elementos que podem ser acessados usando um índice, e (2) aceitam duplicatas.
- **Set**
  Interface para coleções que **não permitem elementos duplicados**.
- **Queue**
  Interface para coleções que armazenam um grupo de elementos em uma ordem específica, comumente em ordem **primeiro a entrar, primeiro a sair (FIFO)**.
- **Map**
  Interface para coleções cujos elementos são armazenados como **pares chave/valor**.

Dessas quatro últimas, **Map** é a única que **não implementa** nem Collection nem Iterable, mas ainda assim é considerada uma coleção porque **contém um grupo de elementos**.

List

Uma List é uma **coleção ordenada** que aceita **elementos duplicados** e permite **acesso por índice**.

A implementação mais comum é ArrayList.

```java
List<String> lista = new ArrayList<>();
lista.add("a");
lista.add("b");
lista.add("a");
```

Você pode acessar e modificar os elementos com:

```java
System.out.println(lista.get(0)); // "a"
lista.set(0, "z"); // substitui "a" por "z"
```

O método indexOf() retorna o índice do **primeiro elemento encontrado** (ou -1 se não encontrar):

```java
System.out.println(lista.indexOf("a")); // 2
System.out.println(lista.indexOf("b")); // 1
System.out.println(lista.indexOf("x")); // -1
```

O método remove() pode remover por **índice** ou por **objeto**:

```java
lista.remove(1); // remove o elemento no índice 1
lista.remove("a"); // remove o primeiro "a"
```

Outros métodos úteis incluem:

```java
lista.size();     // quantidade de elementos
lista.isEmpty();  // true se a lista estiver vazia
lista.contains("a"); // true se a lista contém "a"
```

Você também pode inicializar uma lista com elementos assim:

```java
List<String> lista = Arrays.asList("um", "dois", "três");
```

Atenção: a lista retornada por Arrays.asList() **tem tamanho fixo** — você **não pode adicionar ou remover** elementos, apenas modificar os existentes.

Set

Um Set é uma coleção que **não permite elementos duplicados**.

As duas implementações mais comuns são:

- HashSet: não garante ordem.
- TreeSet: mantém os elementos **ordenados** (por ordem natural ou Comparator).

Exemplo com HashSet:

```java
Set<String> set = new HashSet<>();
set.add("um");
set.add("dois");
set.add("um");

System.out.println(set);
```

Saída possível:

```csharp
[um, dois]
```

A ordem **não é garantida**, e o segundo "um" é ignorado porque já existe no Set.

Exemplo com TreeSet:

```java
Set<String> set = new TreeSet<>();
set.add("um");
set.add("dois");
set.add("tres");

System.out.println(set);
```

Saída:

```csharp
[dois, tres, um]
```

Aqui, a ordem é **alfabética**, ou seja, **ordem natural de String**.

Assim como List, os métodos principais são:

```java
set.size();
set.isEmpty();
set.contains("abc");
set.remove("abc");
set.clear();
```

Como Set **não tem índice**, você não pode acessar elementos com get() nem usar set(index, element).

Queue e Deque

Uma **Queue** (fila) é uma coleção usada para armazenar elementos **em ordem**, normalmente **primeiro a entrar, primeiro a sair (FIFO)**.

Uma **Deque** (fila dupla) é uma fila que permite inserção e remoção de elementos **em ambas as extremidades** (pode atuar como fila ou pilha).

A principal implementação é ArrayDeque, que é mais rápida que Stack e LinkedList para uso como pilha ou fila.

Exemplo de fila (Queue):

```java
Queue<String> fila = new ArrayDeque<>();
fila.add("um");
fila.add("dois");
fila.add("três");

System.out.println(fila); // [um, dois, três]
System.out.println(fila.remove()); // remove "um"
System.out.println(fila); // [dois, três]
```

Métodos principais:

- add(): adiciona ao final.
- remove(): remove o primeiro.
- peek(): retorna o primeiro **sem remover**.
- element(): igual ao peek(), mas lança exceção se a fila estiver vazia.
- poll(): como remove(), mas retorna null se estiver vazia.

Exemplo de pilha (Stack usando Deque):

```java
Deque<String> pilha = new ArrayDeque<>();
pilha.push("um");
pilha.push("dois");
pilha.push("três");

System.out.println(pilha); // [três, dois, um]
System.out.println(pilha.pop()); // remove "três"
System.out.println(pilha); // [dois, um]
```

Métodos principais:

- push(): adiciona no topo.
- pop(): remove do topo.
- peek(): olha o topo **sem remover**.

Map

Um Map é uma coleção de **pares chave/valor**.

Diferente de List e Set, um Map **não implementa** a interface Collection, mas ainda é considerado parte da estrutura de coleções do Java.

As implementações mais comuns são:

- HashMap: **sem ordem garantida**.
- TreeMap: mantém as chaves em **ordem crescente (natural)**.

Exemplo com HashMap:

```java
Map<String, Integer> mapa = new HashMap<>();
mapa.put("um", 1);
mapa.put("dois", 2);
mapa.put("tres", 3);

System.out.println(mapa); // ordem imprevisível
System.out.println(mapa.get("dois")); // 2
```

Métodos principais:

- put(k, v): adiciona ou substitui um valor.
- get(k): obtém o valor associado à chave.
- remove(k): remove o par pela chave.
- containsKey(k): verifica se a chave existe.
- containsValue(v): verifica se o valor existe.
- isEmpty(), size(), clear().

Iterando sobre pares chave/valor:

```java
for (Map.Entry<String, Integer> entrada : mapa.entrySet()) {
    System.out.println(entrada.getKey() + " = " + entrada.getValue());
}
```

- entrySet() retorna um conjunto de pares (Map.Entry<K, V>)
- getKey() e getValue() retornam a chave e o valor, respectivamente.

Exemplo com TreeMap:

```java
Map<String, Integer> mapa = new TreeMap<>();
mapa.put("c", 3);
mapa.put("a", 1);
mapa.put("b", 2);

System.out.println(mapa);
```

Saída:

```r
{a=1, b=2, c=3}
```

A ordem das chaves é natural (alfabética para String).

Pontos-chave

- O **Java Collections Framework** fornece interfaces e classes para armazenar e manipular grupos de objetos.
- As principais interfaces de coleção são:
    - List: ordenada, permite elementos duplicados, acesso por índice.
    - Set: não permite elementos duplicados.
    - Queue: elementos ordenados para acesso FIFO.
    - Map: pares chave-valor.
- As implementações mais comuns são:
    - ArrayList para List
    - HashSet e TreeSet para Set
    - ArrayDeque para Queue e Deque
    - HashMap e TreeMap para Map
- A interface Iterable permite uso do laço for-each com Collection.
- List oferece métodos como add(), get(), set(), remove(), indexOf().
- Set não permite duplicatas e não possui acesso por índice.
- Queue oferece métodos como add(), remove(), peek().
- Deque oferece push() e pop() para uso como pilha (stack).
- Map armazena pares chave → valor e oferece put(), get(), entrySet().

Autoavaliação (Self Test)

---

1. Qual das alternativas é verdadeira sobre Set?

A. Permite elementos duplicados
B. Permite acesso por índice
C. Garante ordem dos elementos
D. Não permite elementos duplicados

---

2. Dado:

java Copiar Editar

```java
List<String> lista = new ArrayList<>();
lista.add("a");
lista.add("b");
lista.add("a");
System.out.println(lista.indexOf("a"));
```

Qual é a saída?

A. 0
B. 1

C. 2
D. -1

---

3. Qual das opções lança exceção se a fila estiver vazia?

A. poll()
B. remove()
C. peek()
D. isEmpty()

---

4. Dado:

```java
Deque<String> pilha = new ArrayDeque<>();
pilha.push("um");
pilha.push("dois");
pilha.pop();
System.out.println(pilha);
```

Qual é a saída?

A. [um, dois]
B. [dois, um]
C. [um]
D. [dois]

---

5. Dado:

```java
Map<String, Integer> mapa = new HashMap<>();
mapa.put("a", 1);
mapa.put("b", 2);
mapa.put("a", 3);
System.out.println(mapa.get("a"));
```

Qual é a saída?

A. 1
B. 2
C. 3
D. null